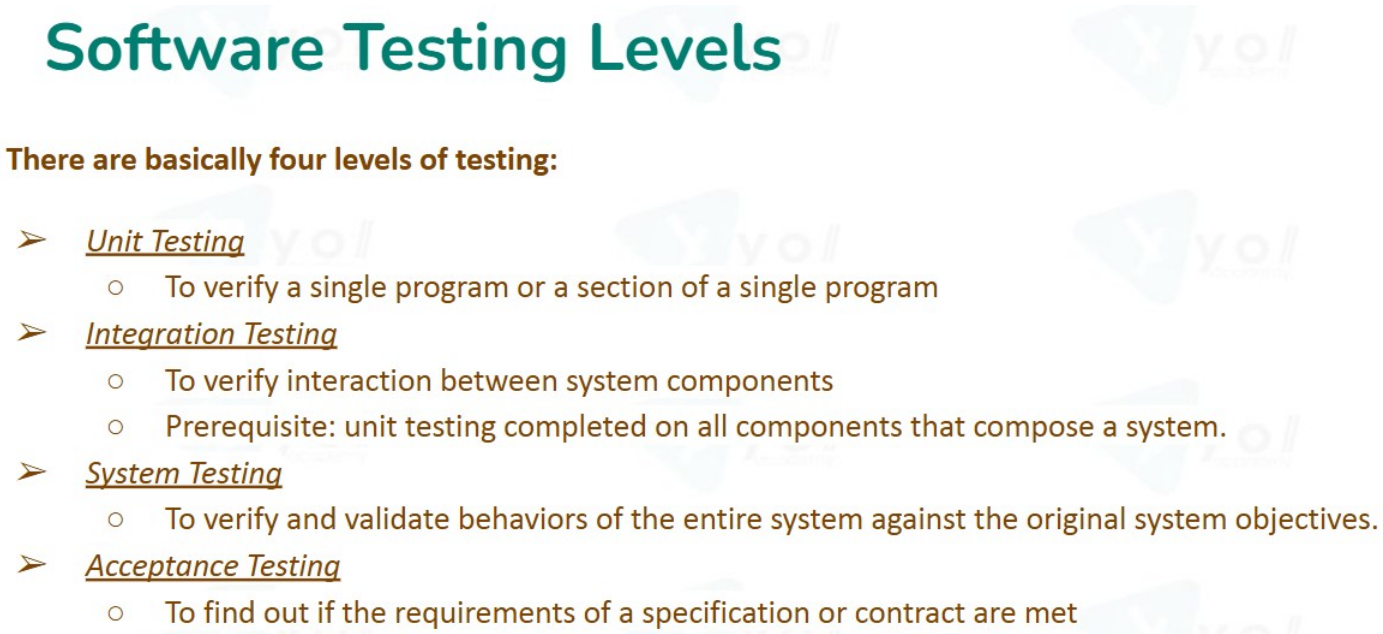

Software Testing Levels
There are basically four levels of testing:
Unit Testing
To verify a single program or a section of a single program
Integration Testing
To verify interaction between system components
Prerequisite: unit testing completed on all components that compose a system.
System Testing
To verify and validate behaviors of the entire system against the original system objectives.
Acceptance Testing
To find out if the requirements of a specification or contract are met

The image describes the **four levels of software testing**: **Unit Testing, Integration Testing, System Testing, and Acceptance Testing**. Below is a detailed explanation of each level with **examples** and **who performs the tests**.

---

1. Unit Testing

- **Purpose**: To test individual units (functions, classes, or modules) of the software to ensure they work correctly in isolation.
- **Who Performs It?** Developers
- **Example**:
  - A function in an e-commerce website calculates the total price of items in a cart.
  - Unit testing ensures that calculateTotal(cartItems) correctly sums up the prices.

---

2. Integration Testing

- **Purpose**: To test the interaction between integrated components or modules to ensure they work together as expected.
- **Who Performs It?** Developers or Testers (depending on the approach)
- **Example**:
  - The e-commerce website integrates the calculateTotal() function with the **payment gateway API**.

- Integration testing ensures that after checkout, the correct amount is passed to the payment system.

---

3. System Testing

- **Purpose**: To test the entire system as a whole to verify that it meets functional and non-functional requirements.
- **Who Performs It?** QA Testers
- **Example**:
  - The e-commerce website is tested end-to-end: from user login, product search, cart management, checkout, and payment processing.
  - System testing checks that all workflows function as expected in a real-world scenario.

---

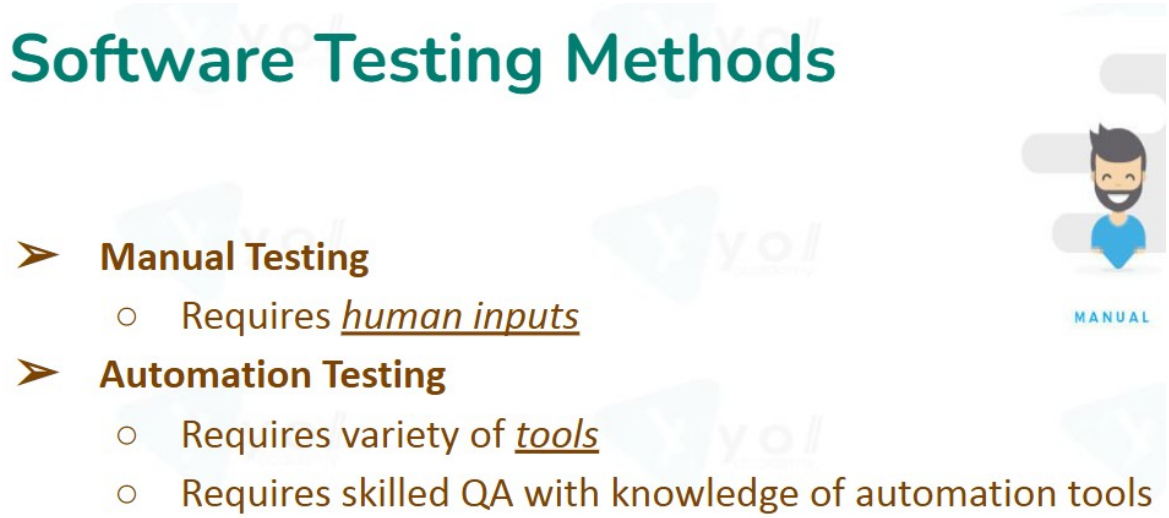
4. Acceptance Testing

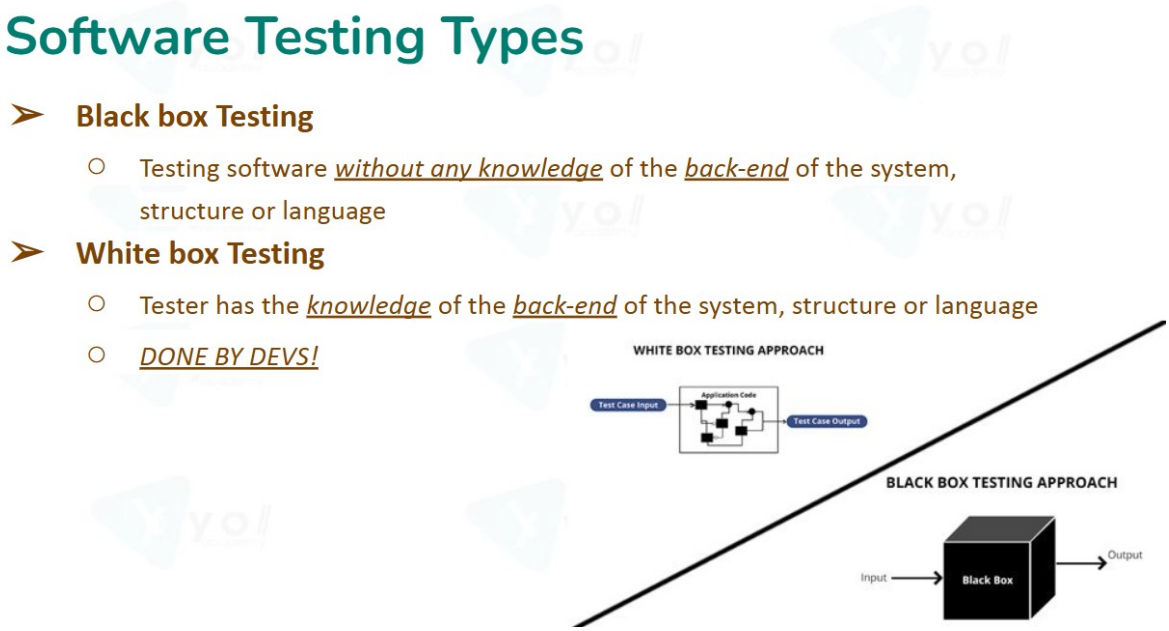
- **Purpose**: To validate whether the software meets business requirements and is ready for deployment.
- **Who Performs It?** End-users, Clients, or Business Analysts
- **Example**:
  - The client of the e-commerce website tests whether customers can successfully purchase items, apply discount codes, and receive confirmation emails.
  - If everything works according to the agreed specifications, the software is approved for release.

Summary: Who Performs Each Test?

Testing Level	Who Performs It?
Unit Testing	Developers
Integration Testing	Developers / Testers
System Testing	QA Testers
Acceptance Testing	End-users / Clients

Each level ensures a different aspect of software quality, helping catch and fix bugs before deployment.




Software Testing Types
Black box Testing
Testing software without any knowledge of the back-end of the system, structure or language
White box Testing
Tester has the knowledge of the back-end of the system, structure or language
DONE BY DEVS!
WHITE BOX TESTING APPROACH
Test Case Input
Application Code
Test Case Output
BLACK BOX TESTING APPROACH
Input
Black Box
Output

1. Black Box Testing

- **What is it?**
  - Testing software **without knowledge** of the backend, structure, or programming language.
  - Focuses on inputs and expected outputs without analyzing internal code.
- **Example:**
  - Testing a **login page** by entering valid and invalid credentials and checking if authentication works correctly.
  - Clicking a **"Buy Now"** button on an e-commerce site to check if it redirects correctly.

- **Who Performs It?**
  - **QA Testers, Manual Testers, or Automation Engineers**
  - Tools like **Selenium, Postman (API testing), or TestComplete** can be used.

---

2. White Box Testing

- **What is it?**
  - Testing with **full knowledge** of the system's backend, structure, and code.
  - Ensures the **internal logic, code flow, and algorithms** work correctly.
- **Example:**
  - **Unit Testing:** A developer writes test cases to check if a **function calculating discounts** gives the correct output.
  - **Code Coverage Testing:** Ensuring all **"if-else"** conditions in a banking application have been tested.
- **Who Performs It?**
  - **Developers and Software Engineers** (since it requires coding knowledge).
  - Tools like **JUnit, NUnit, PyTest, or SonarQube** are used.

➢ **End to End Testing (E2E)**

- Testing a complete application in a situation that simulates what a real user scenarios look like from start to finish

Note: Difference between system testing and E2E testing is that E2E is from one point to another and finds issues in the flow, while system testing is general testing of the application's components

Here's a detailed explanation of **Black Box Testing, White Box Testing, and End-to-End (E2E) Testing**, with **examples** and **who performs them**.

End-to-End Testing (E2E)

- **What is it?**
  - Testing a complete application **from start to finish**, simulating a real user experience.

- Ensures different modules **work together as expected**.

- **Example:**
  - **E-commerce Checkout Process:**
    1. User logs in.
    2. Searches for a product.
    3. Adds it to the cart.
    4. Completes payment.
    5. Receives a confirmation email.
  - **Banking App Transaction:**
    1. User logs in → Transfers money → Gets confirmation SMS.
- **Who Performs It?**
  - **QA Testers, Business Analysts, and Automation Engineers**
  - Automated E2E tools like **Cypress, Selenium, TestCafe, or Playwright** are often used.

Key Differences:

Testing Type	Focus	Who Performs It?	Example
Black Box Testing	Tests without knowing the code	QA Testers, Manual Testers	Checking if a "Reset Password" feature works
White Box Testing	Tests with full code knowledge	Developers, Software Engineers	Checking if a sorting algorithm functions correctly
End-to-End (E2E) Testing	Simulates a real user flow	QA Engineers, Business Analysts	Testing the full order process in an online store

Each testing method is **essential for software quality assurance** and ensures a seamless user experience.

Software Testing Types

- **System Integration Testing**
  - Testing of Software components that have been distributed across multiple platforms
- **Functional Testing**
  - Verifying that a module functions as stated

- **Adhoc Testing**
  - Testing without a formal Test Plan
  - If testing occurs very late in the development cycle, this will be the only kind of testing that can be performed
  - Usually done by skilled testers
- **Shake out testing**
  - A quick test to make sure system is up and running
- **User Acceptance Testing**
  - Testing the system with the intent of confirming readiness of the product and customer acceptance.
- **Smoke testing**
  - test conducted before the delivery and after complete testing
- **Regression testing**
  - Testing to determining if bug fixes have been successful and have not created any new problems.
- **Load Testing**
  - Testing with the intent of determining how well the product handles competition for system resources. The competition may come in the form of network traffic, CPU utilization or memory allocation.
- **Stress Testing**
  - Testing done to evaluate the behavior when the system is pushed beyond the breaking point. The goal is to expose the weak links and to determine if the system manages to recover gracefully.
- **Performance Testing**
  - Testing with the intent of determining how efficiently a product handles a variety of events. Automated test tools geared specifically to test and fine-tune performance are used most often for this type of testing.
- **Some other testing types**
  - Usability Testing
  - Installation Testing
  - Security Testing
  - Penetration Testing
  - Alpha Testing and Beta Testing

Software Testing Types, along with **examples** and **who performs the tests**:

---

1. System Integration Testing (SIT)

- **What is it?**
  - Tests how different software components interact when distributed across multiple platforms.
  - Ensures **modules, APIs, databases, and third-party services** work together.
- **Example:**
  - A banking app integrates with third-party payment gateways (PayPal, Stripe). SIT ensures transactions process correctly.
- **Who Performs It?**
  - **QA Engineers** or **Software Developers** (with integration expertise).

---

2. Functional Testing

- **What is it?**
  - Verifies if a module **performs as expected** based on requirements.
- **Example:**
  - Testing a **search bar** in an e-commerce app to ensure it returns accurate product results.
- **Who Performs It?**
  - **Manual Testers** or **Automated Testing Engineers** using tools like Selenium.

---

3. Adhoc Testing

- **What is it?**
  - Unplanned testing **without a formal test plan** to find unexpected bugs.
  - Usually performed when **time is limited**.
- **Example:**
  - Clicking random buttons in a mobile app to check for unexpected crashes.
- **Who Performs It?**
  - **Experienced Testers** or **Developers** doing exploratory testing.

---

4. Shakeout Testing

- **What is it?**
  - A quick test to **verify basic system functionality** before proceeding with detailed testing.
- **Example:**
  - After deploying a new software build, testers check if the system starts without errors.
- **Who Performs It?**
  - **QA Engineers** or **DevOps Teams** after deployments.

---

5. User Acceptance Testing (UAT)

- **What is it?**
  - Validates if the software **meets business requirements** and is ready for release.
- **Example:**
  - A business team testing an ERP system to ensure it handles invoices correctly before deployment.
- **Who Performs It?**
  - **End Users, Business Analysts, and Clients**.

---

6. Smoke Testing

- **What is it?**
  - Basic testing to check if the major functions work before deep testing.
- **Example:**
  - Testing if a newly deployed website loads, and key buttons respond before doing deeper tests.
- **Who Performs It?**
  - **QA Testers** or **Developers** after a new build is deployed.

What is a Smoke Test?

A **Smoke Test** is a preliminary test conducted to check whether the basic functionalities of a software application are working before proceeding with more in-depth testing. It

ensures that the critical features of the application are functional and stable enough for further testing.

The term **"smoke testing"** originated from hardware testing, where a device was considered a pass if it did not produce smoke when powered on.

Key Characteristics of Smoke Testing

- It is a **shallow and wide** approach to testing.
- It is **not exhaustive**; it focuses on verifying essential functionality.
- Conducted at the **initial stages** after a build is deployed.
- Used to **detect showstoppers** (critical issues that block further testing).
- Can be **automated or manual**.

Examples of Smoke Testing

1. E-Commerce Website Smoke Test

A new version of an e-commerce website is deployed. The testers check:
 Login functionality – Users can log in with valid credentials.
 Product search – Users can search and see product listings.
 Add to cart – Items can be added to the shopping cart.
 Checkout process – Users can enter payment details and proceed.
 Logout function – Users can log out successfully.

If any of these basic functionalities fail, the build is rejected, and further testing is halted until the issue is fixed.

---

2. Mobile Banking App Smoke Test

A banking app releases a new update. The smoke test includes:
 Login with username and password
 Checking account balance display
 Making a small transfer between accounts
 Logging out of the app safely

If the app crashes or the transactions fail at this stage, deeper testing is stopped, and the build is sent back for fixes.

---

3. CRM Software Smoke Test

A CRM (Customer Relationship Management) tool is deployed for testing. A tester verifies:
 User login and dashboard loading

 Creating a new customer record
 Searching for existing customers
 Logging out of the system

If the system fails at login or data retrieval, the build is considered unstable and sent back to developers.

When is Smoke Testing Performed?

- After a **new build** is deployed to the test environment.
- Before running **detailed functional or regression tests**.
- When a **critical fix** or **patch** is deployed.

---

7. Regression Testing

- **What is it?**
  - Ensures new updates **do not break existing features**.
- **Example:**
  - If a new login feature is added, testers check whether previous user profile functionalities still work.
- **Who Performs It?**
  - **Automation Testers** (using tools like Selenium, JUnit) or **Manual Testers**.

What is Regression Testing?

Regression Testing is a type of software testing performed to ensure that recent code changes have not adversely affected the existing functionality of the application. It verifies that previously working features still work after updates, bug fixes, or enhancements.

Key Characteristics of Regression Testing

 Ensures stability – Confirms that new changes do not break existing features.
 Covers impacted areas – Focuses on the modified modules and their dependencies.
 Can be automated – Often automated for efficiency, especially in large applications.
 Repeated frequently – Done after every major or minor code change.
 Includes functional and non-functional tests – Checks both visible features and performance.

When is Regression Testing Performed?

- After a **new feature is added** to the application.
- After **bug fixes** or security patches.
- After **performance improvements** or optimizations.
- Before a **major release or deployment to production**.

Examples of Regression Testing

1. E-Commerce Website Regression Test

A new feature, **"Wishlist"**, is added to an online shopping site. The testers perform regression testing on:
 Login/logout functionality (ensure users can still access their accounts).
 Search and filtering features (verify search results are accurate).
 Shopping cart and checkout process (confirm that adding/removing items works).
 Payment processing (test different payment methods to ensure they work as before).

Even though only the "Wishlist" feature was added, **existing critical functionalities must still work**.

---

2. Banking App Regression Test

A bank fixes a bug in its **mobile deposit feature**. Regression testing ensures that:
 Users can still log in securely.
 Account balances display correctly.
 Fund transfers work as expected.
 ATM withdrawal limits are enforced properly.

Since the deposit module is linked to transactions, testing other financial functions ensures the system remains intact.

---

3. Ride-Sharing App Regression Test

A ride-sharing app updates its **driver rating system**. The testers check:
 User login and profile updates.
 Ride booking and cancellation processes.
 Fare calculations remain accurate.
 Payment and refund processes function correctly.

Even though only ratings were modified, **other connected features must still work as expected**.

---

8. Load Testing

- **What is it?**
  - Tests how the system performs under **high user traffic or data loads**.
- **Example:**
  - Simulating **1,000+ users** accessing an online shopping site during a Black Friday sale.
- **Who Performs It?**
  - **Performance Testers using tools like JMeter, LoadRunner**.

---

9. Stress Testing

- **What is it?**
  - Pushes the system **beyond its limits** to find breaking points.
- **Example:**
  - Sending **millions of requests per second** to a web server to see when it crashes.
- **Who Performs It?**
  - **Performance Testers, DevOps Teams**.

---

10. Performance Testing

- **What is it?**
  - Measures how efficiently a product handles **various operations**.
- **Example:**
  - Checking the **response time of an app** when loading a large dataset.
- **Who Performs It?**
  - **Performance Engineers** using **JMeter, Gatling**.

---

11. Other Testing Types

- **Usability Testing → UX/UI Designers & Testers**
- **Installation Testing → Deployment & DevOps Teams**
- **Security Testing → Cybersecurity Experts**

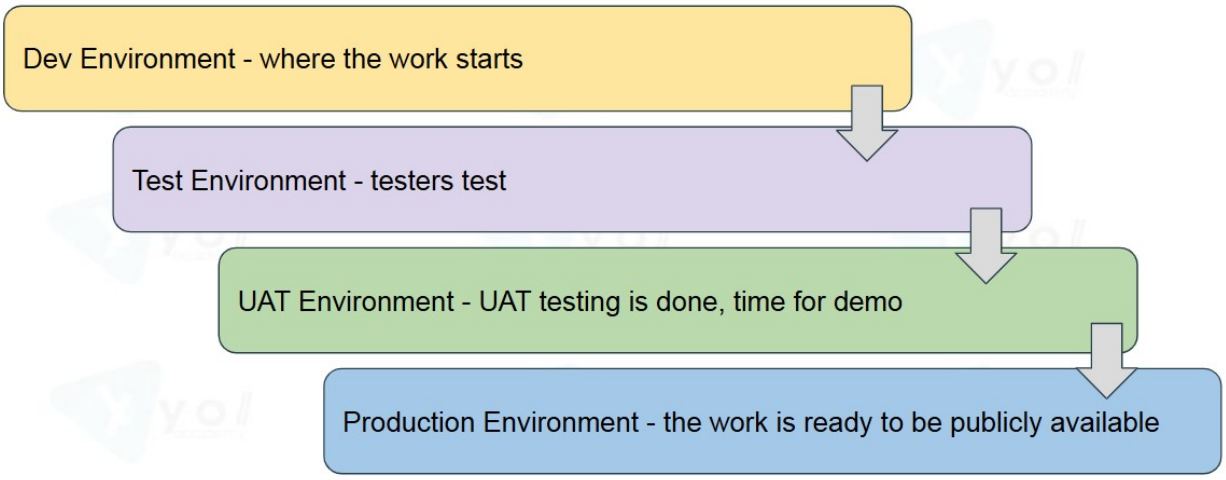
- **Penetration Testing → Ethical Hackers**
- **Alpha & Beta Testing → Internal Users & External Users**

Summary Table

	Who Performs It?	**Example**
System Integration	QA Engineers, Developers	Bank app & PayPal integration
Functional	Manual/Automation Testers	Search bar returns correct results
Adhoc	Experienced Testers	Random clicks in an app
Shakeout	QA Engineers, DevOps	Checking app startup after deployment
User Acceptance (UAT)	Business Users, Clients	Testing an ERP system for invoices
Smoke	QA Engineers, Developers	Checking key functions after a new build
Regression	Automation/Manual Testers	Checking login still works after a UI update
Load	Performance Testers	Simulating 10,000 users on an e-commerce site
Stress	Performance Testers, DevOps	Sending millions of requests to crash a server
Performance	Performance Engineers	Measuring response time of a mobile app
Security & Penetration	Cybersecurity Experts	Checking for SQL injection vulnerabilities
Usability	UX/UI Designers, Testers	Ensuring smooth user experience

Each testing type plays a crucial role in ensuring **software quality, security, and usability**.

Environments


Dev Environment - where the work starts
Test Environment - testers test
UAT Environment - UAT testing is done, time for demo
Production Environment - the work is ready to be publicly available

The **Environments** in software development represent different stages where code is developed, tested, and deployed before being released to end users. Let’s go through each one with **examples**:

---

1. Development (Dev) Environment

- **What is it?**
  - Where developers write, modify, and build code.
  - Code is frequently updated and not stable.
- **Example:**
  - A developer is working on a **new "dark mode" feature** for a mobile app.
  - They write code, run it on their local machine, and test if the UI changes correctly.
  - Bugs are fixed immediately before sending the code for testing.
- **Users:**
  - **Developers**
  - They use **IDEs (e.g., VS Code, IntelliJ, Eclipse)** and **local databases**.

---

2. Test Environment

- **What is it?**
  - A separate environment where QA testers verify the software's functionality.

    - Bugs are identified, logged, and fixed before moving to the next phase.
- **Example:**
    - QA testers check if the **dark mode toggle** works on different browsers and devices.
    - Automated test scripts are run to ensure all UI elements adapt correctly.
- **Users:**
    - **QA Testers, Automation Engineers**
    - Tools like **Selenium, JUnit, Postman (API Testing)** are used.

---

3. User Acceptance Testing (UAT) Environment

- **What is it?**
    - A pre-production environment where **business users or clients** validate the software.
    - Ensures it meets **real-world requirements** before release.
- **Example:**
    - The product owner tests the **dark mode feature** and checks if it meets business needs.
    - A client requests a **different shade of dark background** before approving the release.
- **Users:**
    - **Clients, Business Analysts, Product Owners**
    - Testing is often **manual** with real-world scenarios.

---

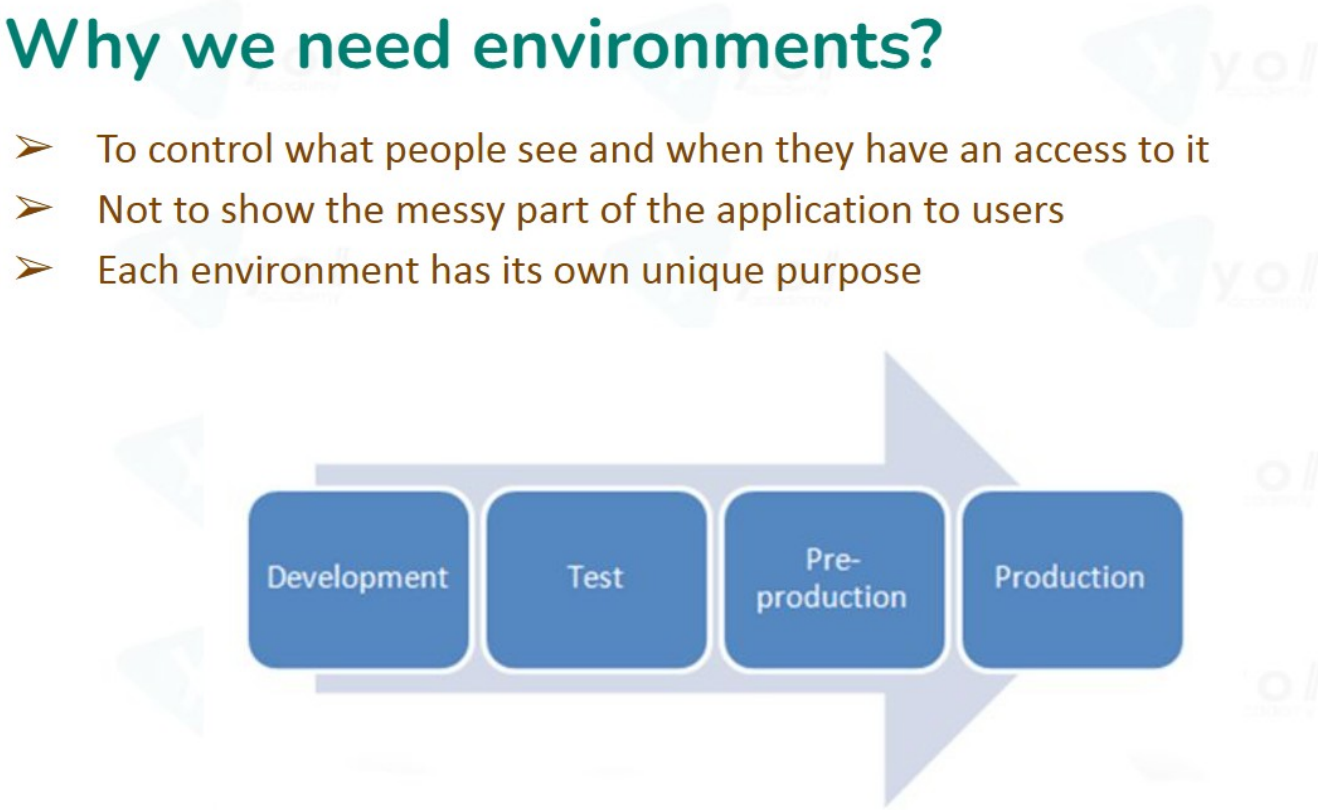
4. Production (Prod) Environment

- **What is it?**
    - The **live environment** where real users interact with the software.
    - Code is fully tested and released for public use.
- **Example:**
    - The **dark mode feature** is now available to all users on the app.
    - Users can switch between light and dark mode seamlessly.
- **Users:**

- **End users, Customers**
- Monitoring tools like **New Relic, Datadog, or AWS CloudWatch** track performance.

Summary Table:

Environment	Purpose	Example	Users
Dev	Developers write and test code	A new feature is coded and tested locally	Developers
Test	QA tests for bugs	Checking if dark mode works on all devices	QA Testers
UAT	Business users verify functionality	Client checks if UI meets expectations	Clients, Business Analysts
Prod	Live system for users	Dark mode is released in the app	End users


Why we need environments?
To control what people see and when they have an access to it
Not to show the messy part of the application to users
Each environment has its own unique purpose
Development
Test
Pre-production
Production

Why We Need Environments?

Environments are necessary in software development to **control visibility, ensure stability, and maintain quality** before releasing software to end users. Each environment serves a specific purpose in the development lifecycle.

---

1. Control What People See and When They Have Access

- **Example:**
  - Developers are building a new **payment gateway feature**.
  - Customers should not see this feature while it is still under development or testing.
  - It remains in **Dev, Test, and Pre-production** before going live.

---

2. Keep the Messy Part Away from Users

- **Example:**
  - A new **mobile app update** introduces a **new UI**.
  - During testing, several **bugs** appear, such as buttons not working.
  - End users should not see these issues, so testing is done in **Test and UAT environments** before moving to production.

---

3. Each Environment Has a Unique Purpose

Environment	Purpose	Example
Development	Code is written and modified	A developer adds a new **search feature**
Test	QA testers verify and find bugs	QA runs automated tests for **login functionality**
Pre-production (UAT)	Business users validate real-world scenarios	The client checks if the **checkout process** works as expected
Production	Live version for users	Customers can now use the **search and checkout features**

Development/Test Environment
Used to build an application
Users and customers cannot access anything
Developers develop, Testers test

Development/Test Environment - Explained with Examples

The **Development/Test environment** is crucial in the software development lifecycle. It is where developers write code, test features, and debug issues before moving forward.

Key Points:

1. **Used to Build an Application**
    - Developers write, modify, and improve code here.
    - Early-stage testing is performed before deployment to other environments.
    - **Example:** A new **user registration feature** is developed and tested before release.
2. **Users and Customers Cannot Access Anything**
    - This environment is **internal** and not accessible to end users.
    - Ensures **bugs and unfinished features** are not exposed.
    - **Example:** A shopping app's **"wishlist" feature** is in development and not available to customers.
3. **Developers Develop, Testers Test**
    - Developers write code, fix issues, and build new features.
    - Testers run automated/manual tests to find defects.
    - **Example:** After developers create a **password reset feature**, testers ensure it works correctly before moving to the next environment.

Summary:

The **Development/Test environment** is a safe space where new ideas and features take shape. Before any application reaches customers, it is **built, tested, and fixed** here !

UAT Environment?
Exact copy of actual application/ production
Used to test the application as it's in production
Team is doing demo to client in UAT
After UAT, the product releases to production

A **UAT (User Acceptance Testing) Environment** is a pre-production environment that replicates the actual production environment to allow end-users or clients to validate the application before it goes live.

Explanation:

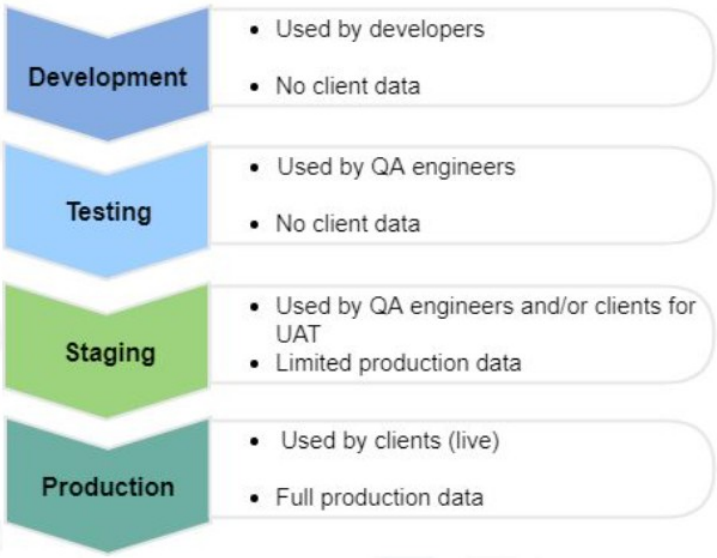
- **Exact Copy of Production:** The UAT environment mimics the production setup, including the same configurations, databases, and integrations.
- **Testing as in Production:** It ensures that the application behaves the same way it would when deployed to real users.
- **Client Demo:** The development team may use the UAT environment to showcase the application's functionality to stakeholders.
- **Final Step Before Production:** Once UAT is successfully completed and approved, the application is deployed to production for actual users.

Examples:

1. **E-commerce Website Testing:**
    - An online shopping platform sets up a UAT environment to test new features (e.g., a new payment gateway). The business users place test orders to verify that everything works smoothly before making it live.
2. **Banking Application:**
    - A bank introduces a new mobile banking feature. Before making it available to customers, internal users test it in the UAT environment to ensure transactions, statements, and notifications work correctly.
3. **ERP System Upgrade:**
    - A company upgrading its ERP system first deploys the new version in UAT. Employees from different departments use it to check if business processes are working correctly before the final rollout.

Production Environment

- Last environment
- Product is ready to be publicly available
- End users can start to use and interact with an application



In software development, there are typically four main environments: **Development**, **Testing**, **Staging**, and **Production**. Each environment serves a specific purpose in the software development lifecycle. Here's an explanation of each environment with examples:

1. **Development Environment**:
   - **Purpose**: This is where developers write, build, and test their code. It is the first environment where the application starts to take shape.
   - **Characteristics**: It is isolated from client data and is used solely by developers.
   - **Example**: A team of developers working on a new feature for a mobile app uses the development environment to write code and perform initial tests. They might use dummy data to simulate user interactions.
2. **Testing Environment**:
   - **Purpose**: This environment is used by Quality Assurance (QA) engineers to test the application for bugs, performance issues, and other defects.
   - **Characteristics**: Like the development environment, it does not contain client data. It is used to ensure the application meets the required quality standards.
   - **Example**: QA engineers test a web application's login functionality to ensure it works correctly under various scenarios, such as incorrect passwords or network issues.
3. **Staging Environment**:
   - **Purpose**: This environment is used for final testing before the application goes live. It often involves User Acceptance Testing (UAT) where clients or end-users validate the application.
   - **Characteristics**: It closely mimics the production environment and may use limited production data to simulate real-world conditions.
   - **Example**: Before launching a new e-commerce website, the staging environment is used to perform end-to-end testing, including payment processing and order fulfillment, to ensure everything works as expected.

4. **Production Environment**:
    - **Purpose**: This is the live environment where the application is publicly available and used by end-users.
    - **Characteristics**: It contains full production data and is the final version of the application that users interact with.
    - **Example**: After thorough testing and approval, a banking application is deployed to the production environment, allowing customers to perform transactions, check balances, and use other features in real-time.

In summary, these four environments ensure that the application is developed, tested, and validated in a controlled manner before being released to the public. Each environment plays a crucial role in delivering a high-quality, reliable product to end-users.

Explanation with Examples:

1. E-Commerce Website

Development Stage: Developers build features like search filters, shopping carts, and payment gateways.
Testing Stage: QA engineers test functionalities like order placement, payments, and login authentication using test data.
Staging Stage: A few internal testers and stakeholders perform final validation with limited production data.
Production Stage: The website goes live, and real customers browse, purchase, and make payments using actual credit cards.

---

2. Banking Application

Development Stage: Developers write code for fund transfers, bill payments, and account statements.
Testing Stage: Testers validate transactions using dummy accounts.
Staging Stage: Limited real transactions are tested by a selected group of employees.
Production Stage: The app is available to real users for financial transactions, withdrawals, and fund transfers.

---

3. Ride-Sharing App

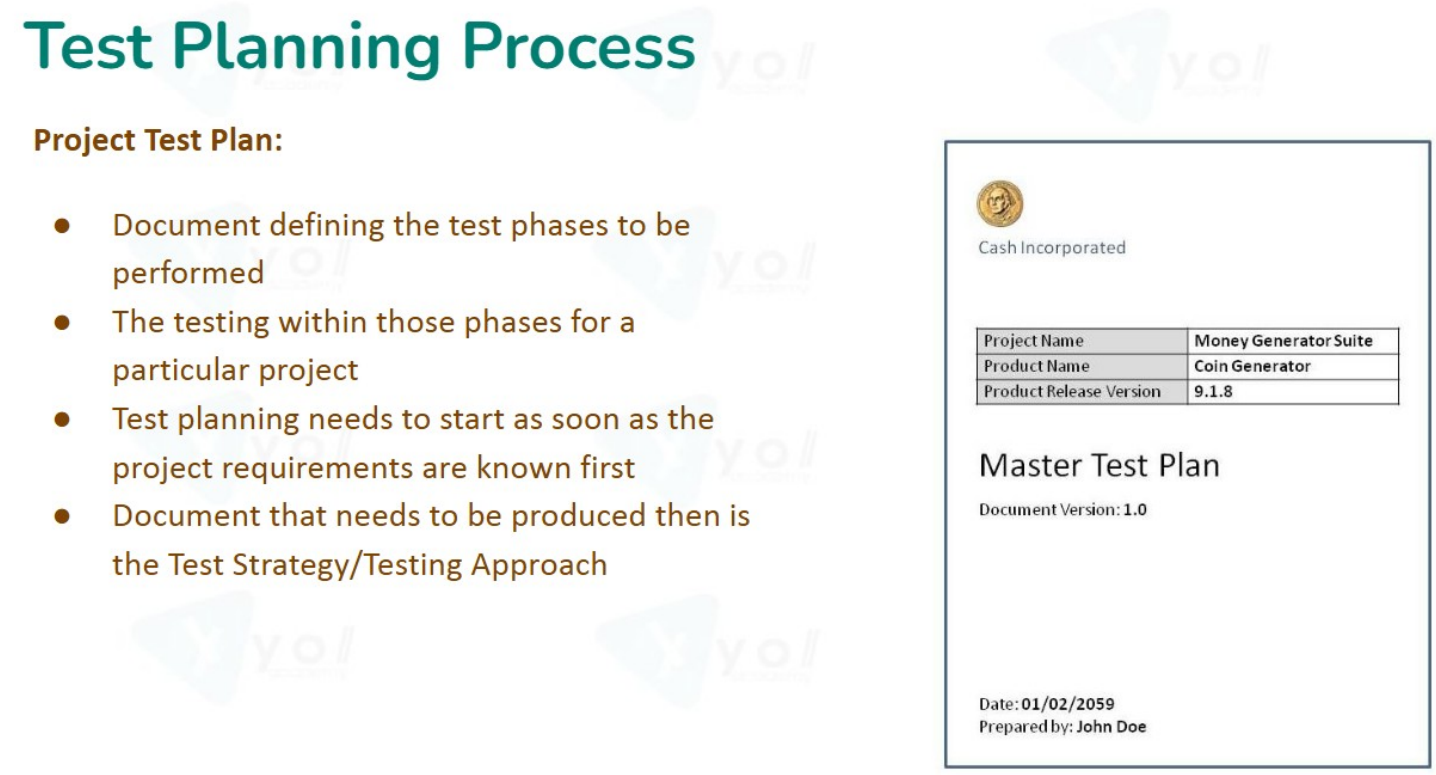
Development Stage: Engineers develop features like ride-booking, fare calculation, and driver tracking.
Testing Stage: QA teams test these features using test accounts and fake ride requests.
Staging Stage: Selected users (internal employees) test the system before launch.
Production Stage: The app is available to the public, allowing real riders and drivers to book and complete trips.

Difference Between Environments in the SDLC

Environment	Purpose	Who Uses It	Data Type	Example
Development	Feature development	Developers	No client data	Writing and debugging new code
Testing	Functional & regression testing	QA engineers	No client data	Testing login, payments, and navigation
Staging	Pre-release testing	QA, stakeholders, and selected users	Limited production data	Final validation before going live
Production	Live application	End users (customers)	Full production data	Customers placing orders and making payments

Conclusion

The **Production Environment** is where software is finally used by actual users. It must be stable, secure, and high-performing. Any issues here can directly impact customers and businesses, so rigorous testing in **Development, Testing, and Staging** phases is essential.


Test Planning Process
Project Test Plan:
Document defining the test phases to be performed
The testing within those phases for a particular project
Test planning needs to start as soon as the project requirements are known first
Document that needs to be produced then is the Test Strategy/Testing Approach
Cash Incorporated
Project Name
Money Generator Suite
Product Name
Coin Generator
Product Release Version
9.1.8
Master Test Plan
Document Version: 1.0
Date: 01/02/2059
Prepared by: John Doe

Test Planning Process - Explained with Examples

What is a Test Plan?

A **Test Plan** is a document that outlines the strategy, scope, resources, and schedule for testing activities in a software development project. It ensures that testing is conducted in a structured and efficient manner.

Key Components of a Test Plan:

 Test Phases – Defines the different testing stages (Unit, Integration, System, UAT).
 Scope of Testing – Specifies what features and functionalities will be tested.
 Test Strategy – Describes the testing approach (manual vs. automation).
 Resources Required – Lists tools, testers, and environments needed.
 Schedule & Milestones – Defines timelines and deadlines for testing.
 Defect Management – Specifies how issues will be tracked and resolved.

Example 1: Test Plan for an E-Commerce Website

Project: Online Shopping Platform

1**Test Phases:**

- **Unit Testing:** Developers test individual components like login, search, and cart.
- **Integration Testing:** Ensures that different modules (payment, checkout, order confirmation) work together.
- **System Testing:** End-to-end testing of the website's full functionality.
- **User Acceptance Testing (UAT):** Selected customers test the website before launch.

2 **Scope of Testing:**

- Validate login/logout functionality.
- Check product search and filtering.
- Ensure that orders are placed correctly.
- Verify payment gateway integration.

3 **Test Strategy:**

- **Manual Testing:** UI/UX validation, exploratory testing.
- **Automation Testing:** Selenium is used for repetitive test cases like login and checkout.

4 **Resources Required:**

- **Testers:** 2 QA engineers, 1 automation tester.
- **Tools:** Selenium, JIRA for bug tracking, TestRail for test case management.

5 **Schedule & Milestones:**

- Test planning: 1 week
- Unit testing: 2 weeks
- System testing: 3 weeks
- UAT: 1 week

Example 2: Test Plan for a Banking Application

Project: Mobile Banking App

1 **Test Phases:**

- **Unit Testing:** Individual banking features like balance inquiry and transactions are tested.
- **Integration Testing:** Ensures funds transfer, bill payments, and account updates work correctly together.
- **System Testing:** Comprehensive end-to-end testing of mobile app functionality.
- **UAT:** A small group of bank employees test real transactions before launch.

2 **Scope of Testing:**

- Verify secure login and authentication.
- Ensure fund transfers between accounts are processed correctly.
- Validate transaction history and statements.
- Check ATM withdrawal requests and notifications.

3 **Test Strategy:**

- **Manual Testing:** Security testing, UI testing, functional testing.
- **Automation Testing:** Regression tests for fund transfer, bill payments using Appium.

4 **Resources Required:**

- **Testers:** 3 QA engineers, 1 security tester.
- **Tools:** Appium, JMeter for performance testing, JIRA for bug tracking.

5 **Schedule & Milestones:**

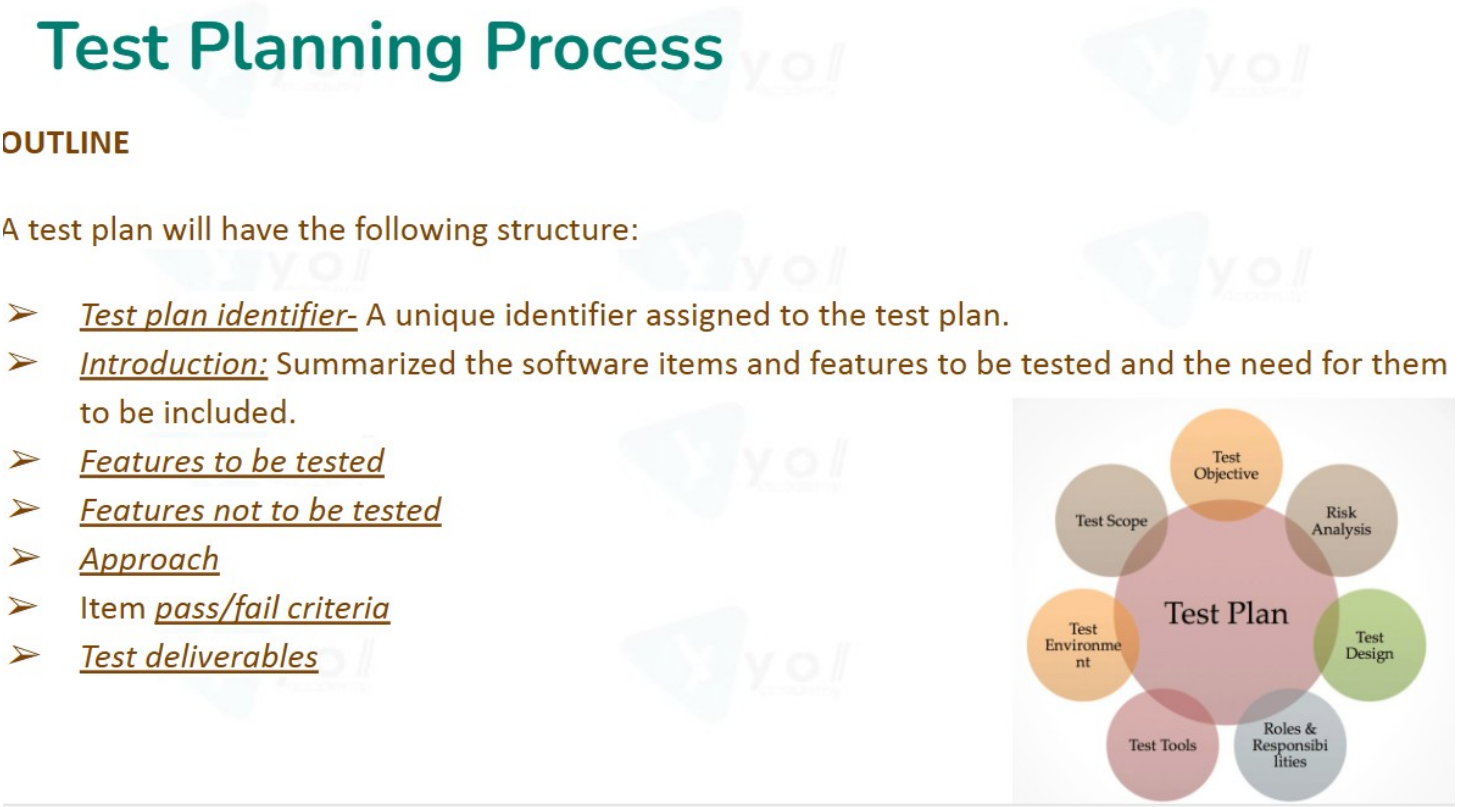
- Security testing: 2 weeks
- Functional testing: 3 weeks
- UAT: 2 weeks

Conclusion

A **Test Plan** is an essential part of software testing, ensuring that testing is well-organized and effective. It helps teams track progress, manage risks, and deliver a high-quality product.

Sample of TEST PLAN :

Live_Project_Test_Plan_SoftwareTestingHelp.pdf


Test Planning Process
OUTLINE
A test plan will have the following structure:
➢ Test plan identifier- A unique identifier assigned to the test plan.
➢ Introduction: Summarized the software items and features to be tested and the need for them to be included.
➢ Features to be tested
➢ Features not to be tested
➢ Approach
➢ Item pass/fail criteria
➢ Test deliverables
Test Objective
Test Scope
Risk Analysis
Test Plan
Test Environment
Test Design
Test Tools
Roles & Responsibilities

The **Test Planning Process** is a structured approach to ensure that software testing is thorough, organized, and effective. It involves creating a detailed test plan that outlines the scope, objectives, resources, schedule, and deliverables of the testing activities. Here's an explanation of the key components of a test plan with examples:

1. **Test Plan Identifier**:
    - **Purpose**: A unique identifier to distinguish the test plan from others.
    - **Example**: "TP_2023_LoginModule" could be the identifier for a test plan focusing on the login module of an application.
2. **Introduction**:
    - **Purpose**: Provides an overview of the software items and features to be tested and explains why they need to be tested.
    - **Example**: "This test plan covers the login functionality of the XYZ application, ensuring secure and efficient user authentication."
3. **Features to be Tested**:

    - **Purpose**: Lists the specific features or components that will be tested.
    - **Example**: Features to be tested might include "User login, Password recovery, and Multi-factor authentication."
4. **Features Not to be Tested**:
    - **Purpose**: Identifies features or components that are out of scope for the current testing phase.
    - **Example**: "The payment gateway integration will not be tested in this phase as it is still under development."
5. **Approach**:
    - **Purpose**: Describes the overall strategy and methods for testing.
    - **Example**: "We will use both manual and automated testing methods. Automated scripts will be created for regression testing, while manual testing will focus on exploratory and usability testing."
6. **Item Pass/Fail Criteria**:
    - **Purpose**: Defines the criteria that determine whether a test item has passed or failed.
    - **Example**: "A test case for login functionality will pass if the user can log in within 2 seconds and fail if it takes longer or results in an error."
7. **Test Deliverables**:
    - **Purpose**: Lists the documents and artifacts that will be produced during testing.
    - **Example**: "Test cases, test scripts, test data, test logs, and a final test summary report."

Additional Components:

- **Test Objective**:
    - **Purpose**: Defines the goals of the testing process.
    - **Example**: "The objective is to ensure the login module is secure, user-friendly, and performs well under various conditions."
- **Test Scope**:
    - **Purpose**: Outlines the boundaries of the testing activities.
    - **Example**: "The scope includes testing the login functionality on web and mobile platforms but excludes third-party integrations."
- **Risk Analysis**:
    - **Purpose**: Identifies potential risks and their impact on the testing process.
    - **Example**: "A risk is that the login module may not handle high traffic efficiently, leading to potential downtime."
- **Test Design**:
    - **Purpose**: Involves creating detailed test cases and scenarios based on requirements.
    - **Example**: "Test cases will include valid and invalid login attempts, password recovery, and session timeout scenarios."
- **Tests Performance**:
    - **Purpose**: Execution of test cases and recording of results.
    - **Example**: "Performance testing will simulate 1000 users logging in simultaneously to check system response times."
- **Test Tools**:
    - **Purpose**: Lists the tools that will be used for testing.

    - **Example**: "Selenium for automated testing, JIRA for bug tracking, and LoadRunner for performance testing."
- **Roles & Responsibilities**:
    - **Purpose**: Defines the team members involved in testing and their specific roles.
    - **Example**: "QA Engineer - Execute test cases, Developer - Fix identified bugs, Test Manager - Oversee the testing process."

In summary, the Test Planning Process ensures that all aspects of testing are carefully considered and documented, leading to a more organized and effective testing phase. This structured approach helps in identifying and resolving issues early, ensuring a high-quality software product.

The **Test Planning Process** is a structured approach to ensure that software testing is thorough, organized, and effective. It involves creating a detailed plan that outlines the tasks, resources, schedule, and risks associated with testing. Here’s an explanation of each component with examples:

1. **Testing Tasks**:
    - **Purpose**: Defines the specific activities that need to be performed during the testing phase.
    - **Example**: For a new e-commerce website, testing tasks might include "Create test cases for user registration," "Execute performance tests to check load handling," and "Validate the checkout process."
2. **Environmental Needs**:
    - **Purpose**: Identifies the hardware, software, and network configurations required for testing.
    - **Example**: "The testing environment needs Windows 10, Chrome browser version 85, and a dedicated server with 8GB RAM to simulate real-world conditions."

3. **Responsibilities**:
    - **Purpose**: Outlines the roles and responsibilities of each team member involved in the testing process.
    - **Example**: "QA Engineer - Execute test cases and report bugs, Developer - Fix identified issues, Test Manager - Coordinate testing activities and ensure deadlines are met."
4. **Staffing and Training Needs**:
    - **Purpose**: Identifies the personnel required for testing and any training they might need.
    - **Example**: "We need two additional QA engineers with experience in automated testing tools like Selenium. Training sessions will be conducted to familiarize the team with the new bug tracking tool, JIRA."
5. **Schedule**:
    - **Purpose**: Provides a timeline for the testing activities, including start and end dates for each task.
    - **Example**: "Test case creation - Week 1, Test execution - Weeks 2-3, Bug fixing and retesting - Week 4, Final test report - Week 5."
6. **Risks and Contingencies**:
    - **Purpose**: Identifies potential risks that could impact the testing process and outlines plans to mitigate them.
    - **Example**: "Risk: Delays in bug fixes by the development team. Contingency: Allocate additional resources to the development team or extend the testing timeline by one week."
7. **Approvals**:
    - **Purpose**: Ensures that the test plan is reviewed and approved by relevant stakeholders before execution.
    - **Example**: "The test plan must be approved by the Project Manager, QA Lead, and Client Representative before testing begins."

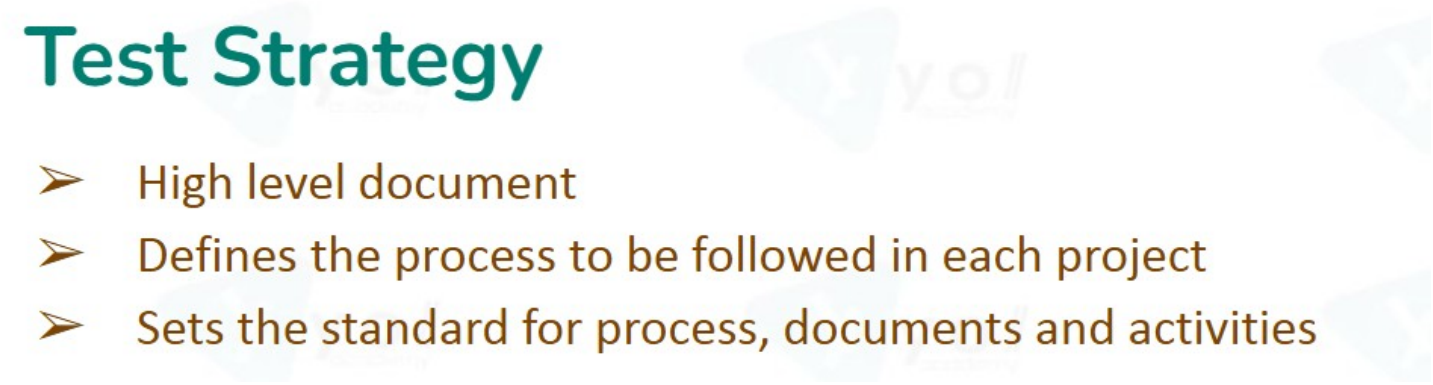
Example Scenario:

Imagine a team is developing a new mobile banking application. The test planning process might look like this:

- **Testing Tasks**: "Create test cases for login functionality, fund transfer, and account balance check."
- **Environmental Needs**: "Testing environment requires iOS and Android devices, various network conditions, and a simulated banking backend."

- **Responsibilities**: "QA Engineer - Execute test cases, Developer - Fix bugs, Test Manager - Monitor progress."
- **Staffing and Training Needs**: "Hire one more QA engineer and provide training on mobile testing tools like Appium."
- **Schedule**: "Test case creation - First week, Test execution - Second and third weeks, Bug fixing - Fourth week."
- **Risks and Contingencies**: "Risk: Payment gateway integration issues. Contingency: Have a backup payment gateway for testing."
- **Approvals**: "Test plan to be approved by the Project Manager and Client before starting."

In summary, the Test Planning Process ensures that all aspects of testing are carefully planned and documented, leading to a more organized and effective testing phase. This structured approach helps in identifying and resolving issues early, ensuring a high-quality software product.


Test Strategy
High level document
Defines the process to be followed in each project
Sets the standard for process, documents and activities

A **Test Strategy** is a high-level document that outlines the testing approach, objectives, resources, and schedule for a project. It sets the standards for processes, documentation, and activities to ensure that testing is consistent and effective across the project. Here's an explanation with examples:

1. **High-Level Document**:
    - **Purpose**: The test strategy provides a comprehensive overview of the testing approach without going into detailed test cases.
    - **Example**: For a mobile banking app, the test strategy might state, "The testing approach will include functional, performance, and security testing to ensure the app is reliable, fast, and secure."
2. **Defines the Process**:
    - **Purpose**: It outlines the specific processes and methodologies that will be followed during testing.

    - **Example**: "We will follow an Agile testing process with iterative testing cycles. Each sprint will include test planning, execution, and review."
3. **Sets Standards**:
    - **Purpose**: Establishes the standards for documentation, test activities, and quality metrics.
    - **Example**: "All test cases must be documented in the Test Management Tool (e.g., JIRA) and follow a standardized template. Test results should be recorded with pass/fail criteria clearly defined."

Example Scenario:

Imagine a project to develop a new e-commerce platform. The test strategy might include the following elements:

- **Objectives**: "Ensure the platform is user-friendly, handles high traffic, and processes payments securely."
- **Scope**: "Testing will cover the user interface, payment gateway, product search, and checkout process."
- **Approach**: "We will use a combination of manual and automated testing. Automated tests will be used for regression testing, while manual tests will focus on usability and exploratory testing."
- **Resources**: "The testing team will consist of 4 QA engineers, 2 automation testers, and 1 test manager. Tools like Selenium for automation and LoadRunner for performance testing will be used."
- **Schedule**: "Testing will be conducted in three phases: Unit testing during development, integration testing after each sprint, and system testing before the final release."
- **Risks and Mitigation**: "Risk: Delays in development may compress the testing timeline. Mitigation: Allocate additional resources and prioritize critical test cases."
- **Deliverables**: "Test plans, test cases, test scripts, test execution reports, and a final test summary report."

In summary, a test strategy provides a clear and structured approach to testing, ensuring that all team members understand the objectives, processes, and standards. This helps in delivering a high-quality product that meets the project requirements and user expectations.

#	Test Plan	Test Strategy
1	A test plan is derived from Software Requirement Specification (SRS), describing in detail the scope of testing and the different activities performed in testing.	A test strategy is a high-level document describing the way testing is carried out.
2	A test plan is project level.	A test stategy organization level
3	It describes the whole testing activities in detail - the techniques used, schedule, resources etc.	It describes the high-level test design techniques to be used, environment specifications etc.
4	It is prepared by test lead or test manager.	It is generally prepared by the project manager.
5	Components:The major components of Test Plan include – Test Plan ID, test environment, features to be tested, entry/exit criteria, status, type of testing, brief introduction etc	The major components of Test Strategy include – Scope, Objective, business issues, risks, testing approach, testing deliverables, defect tracking, training, automation etc.
6	A Test Plan usually exists individually.	Test strategy is divided into multiple test plans that are taken care further independently.